Exercício 02

Observações Gerais

I. A implementação da solução do problema proposto deverá:
   a. Ser realizado em linguagem C/C++;
   b. Ser feita no arquivo com o mesmo nome do exercício. Por exemplo: a solução do exercício 1 será realizada no arquivo exercício_1.cpp; do exercício 2 no arquivo exercício_2.cpp; e assim por diante;

II. Todos os arquivos das implementações dos algoritmos de ordenação, lista linear simples, pilha, fila e árvore de pesquisa, necessários para a solução (conforme é descrito no enunciado) do exercício serão fornecidos;

III. O aluno somente poderá realizar a implementação da solução no arquivo de trabalho do exercício (exercicio_1.cpp, exercicio_2.cpp, etc). Alterações nos outros arquivos não serão consideradas na correção;

Observações sobre a Compilação e Execução do Exercício

I. O programa (solução do exercício) deverá compilar sem erros. Caso existam erros de compilação a solução não será considerada, ou seja, será atribuída nota igual a zero;

II. O arquivo exercício_x.cpp, submetido e que contém a implementação da solução do exercício, será compilado com os arquivos idênticos ao anexo do enunciado do exercício. Por exemplo, o anexo do enunciado contém os arquivos: *listalinearsimples.h, listalinearsimples.cpp, exercício_1.cpp, main_exercicio_1.cpp*. O arquivo *exercício_1.cpp* deverá conter a solução do problema proposto e consequentemente, deverá ser postado como a solução. Durante a correção, o arquivo *exercício_1.cpp* (enviado como solução) será compilado com os arquivos *listalinearsimples.h, listalinearsimples.cpp e main_exercicio_1.cpp* idênticos anexos ao enunciado. Por isso, não altere os arquivos que não contenham a solução do exercício;

III. Uma vez que o programa compile sem erros, ele não poderá apresentar advertências de compilação. Caso existam advertências de compilação, a nota final do exercício será reduzida em 20%;

IV. O programa executável, gerado pela compilação, deverá iniciar sem cancelar sua execução (sem erros de execução SEGMENTATION FAULT, CORE DUMP, entre outros). Caso o programa executável cancele sua execução, a nota final atribuída ao exercício será zero.

Observações sobre o Teste da Solução do Exercício

I. O teste será realizado em duas etapas:
   a. A primeira etapa será um teste realizado com os mesmos dados constantes no enunciado do exercício;
   b. A segunda etapa será um teste realizado com outros dados contendo as mesmas características dos tipos de dados do teste da primeira etapa, porém com uma quantidade de informações diferentes;

Observações sobre Plágios

I. A verificação de plágio de soluções será realizada pela plataforma MOSS (https://theory.stanford.edu/~aiken/moss/);
II. Caso sejam detectados mais do que 80% de similaridade entre as soluções dos alunos, será considerado um plágio e consequentemente a solução não será considerada, ou seja, será atribuída nota igual a zero;

Enunciado

No segundo exercício você deverá desenvolver uma aplicação que ordene um conjunto de dados inteiros cujos valores devem ser maiores e iguais a zero, utilizando o algoritmo BucketSort. A explicação da estratégia de ordenação do bucket sort podem ser encontradas nos seguintes links:

- https://pt.wikipedia.org/wiki/Bucket_sort
- https://www.simplilearn.com/tutorials/data-structure-tutorial/bucket-sort-algorithm

O container deverá conter 1000 baldes (buckets) e cada balde deverá armazenar cem números no máximo. **Não devem existir números repetidos no balde**, ou seja, entradas repetidas devem ser descartadas .

Sendo assim, o balde 0 deverá armazenar números entre 0 e 99. O balde 1 deverá armazenar números entre 100 e 199. O balde 2 deverá armazenar números entre 200 e 299, e assim por diante.

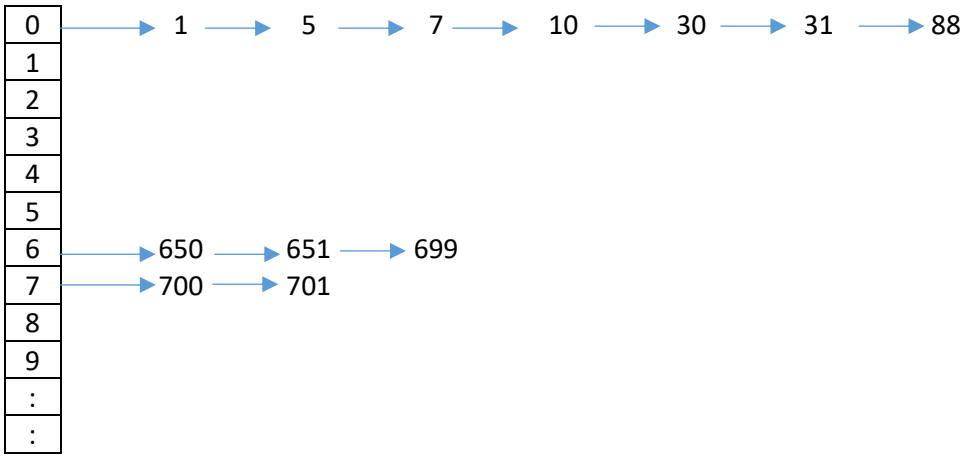
O balde será uma lista linear ordenada.

Observe o exemplo a seguir:

Valores inseridos na lista:

10, 30, 1 ,7, 700, 699, 88, 5, 700, 699, 701, 30, 31, 650, 650, 650, 651

Estado do bucket após a inserção dos valores:



Arquivos anexos ao enunciado:

descriptor.h

descriptor.cpp

exercicio_2.h

exercicio_2.cpp

exercicio_2_main.cpp

OBSERVAÇÃO:

Você devera implementar as seguintes funções no arquivo exercício_2.cpp:

- `void addBucket(DESCRIPTOR bucket[], INFO info)`
- `void insertInOrder(DESCRIPTOR *descriptor, INFO info)`

Apenas o arquivo exercício_2.cpp deverá ser postado, via Teams, como a solução do exercício.